Appeared in the Proceedings of Creativity and Cognition2002, Loughborough, UK, ACM Press, pp.103-110, October, 2002.

Interaction Design as a Collective Creative Process

Kumiyo Nakakoji **Yasuhiro Yamamoto**
PRESTO, JST and RCAST, University of Tokyo
KID (Knowledge Interaction Design) Laboratory
RCAST, University of Tokyo, 4-6-1 Komaba, Meguro,
Tokyo, 153-8904, Japan
+81-3-5452-5286
kumiyo@ai.rcast.u-tokyo.ac.jp yxy@acm.org

Atsushi Aoki
SRA-KTL Inc.
3-12 Yotsuya, Shinjyuku
Tokyo, 160-0004, Japan
+81 3-3357 9011
aoki@sra.co.jp

ABSTRACT
This paper reports our case study on an ongoing interaction-design-centered software development project (ART project) viewed as an evolutionary collective creative process. In this project, a visual interaction designer and a programmer have collaboratively produced a series of software application fragments, which are executable interactive software objects, including a various types of movie players, innovative 3D visualizations and application systems. Visual interaction design is viewed as a process of seeking for compromises between *what are desirable* (expressed by the designer) and *what are possible* (expressed by the programmer). In the collaboration, each of the designer and the programmer collects, represents, interacts with, and reflects on a various types of representations. This paper characterizes the visual interaction design task, presents our framework to analyze the creative processes, and reports in detail how their creative processes have been carried out.

Keywords
Visual interaction design as a creative process, collective creativity, evolutionary creativity, case study, the ART (Amplifying Representational Talkback) concept

INTRODUCTION

In early 2000, a project was formulated to design and develop interactive systems to support experimenters in conducting empirical studies. Since then, the project has shifted its goal to focus more on the design and development of intellectual creative tasks. The project is called the ART project because its visual interaction design is based on the *ART (Amplifying Representaitonal Talkback)* design principle [15]. The ART principle emphasizes the importance of visual interaction and the power of external representations.

In the ART project, an interaction designer and an expert programmer have been intensively collaborating with each other and have produced a number of software application fragments, which are executable interactive software objects, including a various types of movie players, innovative 3D visualizations and ART application systems [8][16][17]. The programs are written in VisualWorks Smalltalk and made publicly available as open source [1].

The goal of this paper is to report not details of these artifacts produced from the ART project, but a case study on this ongoing project viewed as an evolutionary collective creative process. Through interviews both with the designer and the programmer, we have found that each of the designer and the programmer collects, represents, interacts with, and reflects on a variety of representations during the process. An interplay between the representations generated by the designer and the representations generated by the programmer plays a critical role producing creative artifacts.

In what follows, we first give an overview of the ART project and describe how the project has been carried out. We then discuss the nature of visual interaction design and interaction-design-centered software development and how and why collaboration between the designer and the programmer has been critical for the task. We present a process model, which has four facets, consisting of collection, representation, interaction, and reflection, as a framework that characterizes the evolutionary collective creative process. The following section describes what representations each of the designer and the programmer uses in their process.

THE ART PROJECT

A Project Overview: Artifacts

The ART project is based on the concept called ART (Amplifying Representational Talkback) [15]. Based on Donald Schoen's design theory on *reflection-in-action* [10], the project has focused on the role and effects of external representations that play during the user's thinking processes [18]. The ART principle stresses two points: a

C&C'02, October 14–16, 2002, Loughborough, Leic, United Kingdom.

[1] www.kid.rcast.u-tokyo.ac.jp/systems/ARTware

user should be able to externalize what the user wants with as little cognitive load as possible, and a user should be able to perceive what has been externalized with as little cognitive load as possible. The concept embraces the role of perception as a compliment to cognition, and respects the power of paper and pencil. Paper and pencil seldom bothers our thinking processes, and with them, we can externalize what we want to externalize, with various degrees of precision and commitment [3][14]. The goal of the project following the ART concept has been how to go beyond paper and pencil.

The project has so far applied the ART design principle in the design of interactive systems for early stages of linear information design [16][17]. In linear information design, such as collage-style writing, a user needs to construct *parts* and the *whole* by trial-and-error. The parts and the whole depend on each other and coevolve forming a hermeneutic circle. To support linear information design, instead of allowing a user to directly manipulate the linear information, we provide a space for objects as parts framing the whole. A user can freely place objects in a 2D space, and the system automatically serializes the objects from top-to-bottom or left-to-right, whatever accommodates the "natural" order in the application domain [17]. The spatial positioning of objects, a type of spatial hypertext, allows users to externalize a variety of "meta" comments for linear information design by using a variety of visual cues, such as size, distance between objects, or alignment. Figure 1 shows the architecture for this interaction method, which uses spatial hypertext as an instrument for early stages of linear information design, consisting of ElementEditor (EE; for creating an object to be placed in the space), ElementSpace (ES; the space), and DocumentViewer (DV; for showing the serialized information) (Figure 1).

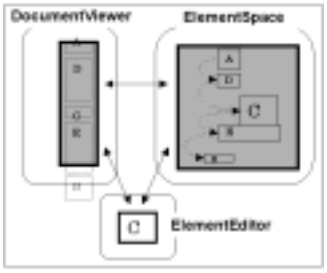

Figure 1: The Architecture for Linear Information Design used in the ART project

Four interactive systems have been designed and developed based on this framework. Figure 2 shows the four systems: ART#001 for collage-style writing, ART#002 for notes-summarization, ART#003 for multimedia data analysis, ART#004 for video editing. They are all for early stages of linear information design using EE, ES, and DV. For instance, in ART#003 a user can view multimedia data (e.g., a subject's video) in EE, and identify an interesting part of the movie. The user can drag and drop the segmented movie and place it in ES. The user may move and resize the object; for instance, one might move important factors toward the top, and make interesting parts but unknown factors larger. The user may textually annotate the positioned object. Each object together with its annotation is serialized (from top-to-bottom or left-to-right as the user specifies) and shown in DV in a table format. The content of DV can be saved in an HTML format.

Figure 2: Four ART Systems Produced based on the Architecture

The Jun-NSN library has evolved as the project produced the four systems. The library is rich in its interaction idioms especially for this particular architecture (Figure 1). For instance, the library has several application fragments to implement a "space" where a user can place multimedia objects (text, movie, sound, image). The project has accumulated multiple code fragments to manipulate the space; for instance, to show a trajectory for an object as the user is dragging it (Figure 3).

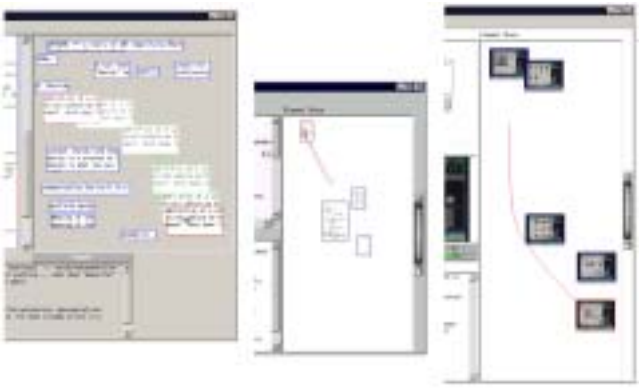

Figure 3: A Variety of Drawing Trajectories Provided by the Library

A Project Overview: Processes

Having visual interaction design as its central focus, the

ART project members consist of an interaction designer, a programmer, and several part-time test users. As a general framework, the project has proceeded by iterating the following:

(1) The interaction designer first draws sketches to illustrate how a system should look and verbally describes to the programmer how a user would interact with the system.

(2) Based on the description provided by the designer, the programmer implements necessary functionality and shows executable programs to the designer.

(3) The designer uses and interacts with the programs, and gives feedback to the programmer.

(4) The programmer fine-tunes the program based on the feedback and makes the program available for test-users, who further give feedback to the designer and the programmer.

The ART project has been going through a cycle of two phases: the *individually working (IW) phase*, which usually lasts for two to three weeks, and the *face-to-face collaboration (FC) phase*, which lasts for about a week (Figure 4). The steps (1) and (2) above mainly take place during the IW phase. The steps (3) and (4) take place during the FC phase.

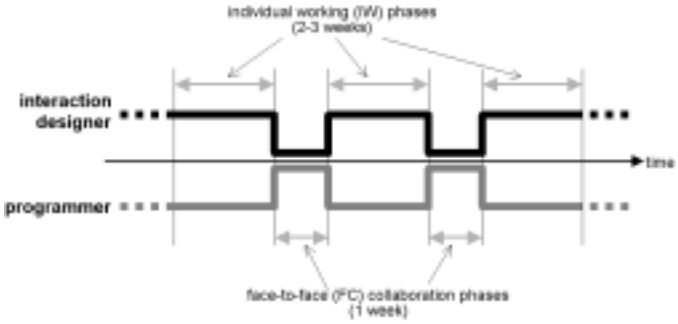

Figure 4: The Two Process Phases in the ART Project

If we look at the process more carefully, however, the process is not as simple as illustrated as the above four steps.

The interaction designer's role was to prioritize what is desirable and to make compromises with what is possible in terms of programming and computational hardware limitations. In doing so, the interaction designer closely collaborates with the programmer in understanding what is possible in terms of what is desirable.

The interaction designer produces sketches and describes to the programmer what the designer thinks is a desirable interaction. The programmer listens to the designer, and implements application fragments, which are executable objects that illustrate computational limitations (such as rendering speed, and algorithms). By using tangible, executable application fragments, the designer and the programmer further discuss what can be and cannot be achieved.

When the designer and the programmer both see that most of the conflicting requirements are resolved and desired functions are achievable, application fragments are integrated within a single window resulting in a prototype system. Test-users then use the system and identify likes and dislikes about the system. Some of their criticisms are taken into account and reflected in the system redesign, but most of them are made further elaborated with the help of the designer, and eventually the test-users often become convinced why such design decisions are made as they are.

Overall, the communication goes both ways. It is not only the designer showing the sketch and the programmer implementing it (Figure 5, top), the programmer sometimes implements "cool stuff" and shows the designer, then the designer tries to understand the implication of the possibility (Figure 5, bottom).

Figure 5: Two Way Designer-Programmer Communication

The next section takes a closer look at this collaboration as a collective creative process.

VISUAL INTERACTION DESIGN IN THE INTERACTION-DESIGN-CENTERED SOFTWARE DEVELOPMENT

From the very beginning, the ART project has focused on visual interaction design. In order for a system to be an effective cognitive tool, visual interaction design plays a critical role [13]. Visual interaction design is neither information visualization nor user interface design. It is about how a user interacts with the system and how the system gives feedback to the user in response to the interaction through a visual representation displayed on the screen.

The ART project we report here has been carried out as an interaction-design-centered software development project [9].

Based on a goal stated at an abstract level, such as "*to produce a movie editing system, which would allow a user to clip a part of the movie through natural interaction*," the interaction designer identifies a set of design requirements based on the ART design principle through a number of sketches of visual appearance for the system. In this process, the designer does not just draw a picture of "the" ideal interface. Rather, the designer tries to take into

account what would be the limitation of the computational processing speed, a screen resolution, or required memory size.

When the designer is not sure about such technical aspects, for instance, how long it would take to extract 20 frames out of the 600x800 30-minute movie, the designer asks the programmer for more information. Design requirements identified in this manner, for instance, "*the system needs to have two scrollbars; one is for the user to see which part of the whole movie is currently clipped, and the other is for the user to focus on the currently clipped part allowing the user to extend or shorten the part by slightly moving a handle on the scrollbar*," are communicated with the programmer through some of the produced sketches, verbal descriptions and gestures.

The programmer then creates necessary object models for achieving the requirements, such as handling a movie or controlling two scrollbars. Detailed design decisions on design features that are necessary to be made in order to program, such as in what timing the system updates the movie view with what movie frame, emerge and are identified during the programming process. For some of the design features, the programmer makes a decision by himself and proceeds the programming task without asking the designer by assuming what the designer would say based on the ART principle. For other design features, however, the programmer pauses his programming task and asks the interaction designer how to decide.

Thus, the visual interaction design task can be viewed as a process of seeking for the "right" balance between *what is desirable* (specified by the designer) and *what is possible* (demonstrated by the programmer) (Figure 5). Because of the limitations imposed by the computational power, expressiveness and resources, some design requirements need to be prioritized and compromised; making design decisions is a process of making priority and seeking for compromises. As a result of making such compromises, design features are implemented and instantiated as executable application fragments. As stated above, when the designer and the programmer both see that most of the conflicting requirements are resolved and desired functions are achievable, application fragments are integrated within a single window resulting in a prototype system (Figure 6).

We view the software produced by the ART project as creative artifacts. They are creative because they represent innovative solutions while providing useful functionality [6]. They are neither a result of the interaction designer's creative process nor that of the programmer's creative process. They are a result of the synergistic collaboration between the designer and the programmer, not necessarily through synchronous collaboration to make a decision, but through mutual respect and complementary knowledge supplement dissolving the issue of symmetry of ignorance [1]. They are a result of interplay between the two stakeholders who have expertise in two different domains: in designing, and in programming.

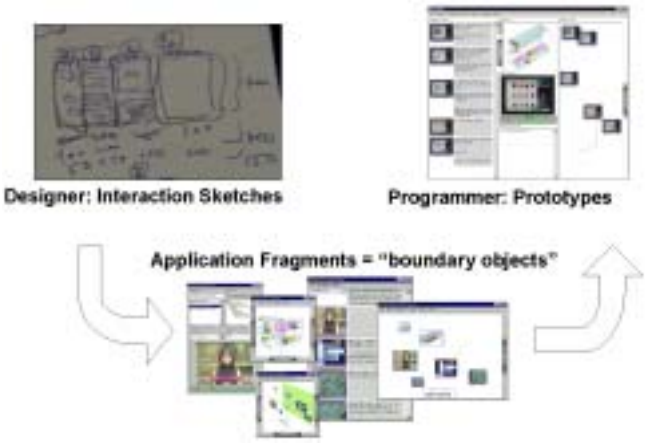

Figure 6: From Sketches to Prototypes via Application Fragments

The next section provides a framework that guides our case study on this creative process.

A PROCESS MODEL FOR COLLECTIVE CREATIVITY

Characterizing The Creative Process

The process followed by the ART project has been neither a *revolutionary* nor *impromptu*, but an *evolutionary* creative process [12]. Designed artifacts emerge as the designer and programmer collaborate with each other over time. Artifacts are produced through a small, incremental process rather than as a historical revolutionary process. Each design decision made during the project is not necessarily a large leap; however, the collection of such small decisions have resulted in quite innovative and useful solutions [6].

Shneiderman [12] differentiates three perspectives on creative people: *inspirationalist*, who depend on informal representations, such as free association, brainstorming, and lateral thinking, *structuralist*, who use more formal, structured representations such as charts, decision trees, or structured diagrams, and *situationalist*, who exhibit their creativity through a social setting working within or across communities of practice. Stakeholders involved in the ART project show a combination of the three perspectives.

The interaction designer demonstrates a quite strong *inspirationalist* perspective. The designer pushes his creative ideas mainly through hand-written sketches and informal visual representations, interacting with emerging meanings from such representations.

In contrast, the programmer demonstrates a *structuralist* perspective. Programming is a structure-oriented process by nature. The programmer needs to deal with the MVC (Model-View-Controller) architecture [5] with which he interacts to uncover tacit requirements and unattended design decisions.

The collaborative aspect between the two people shows the *situationalist* perspective. As discussed above, visual interaction design is a process of seeking for compromises between what is desirable and what is possible. The designer's idea needs to be put in the context of programming through the communication with the programmer. The programmer's design decisions in programming, some of which are not articulated by the designer, have to be evaluated in the eye of the designer to make sure if such decisions are *in harmony with* the rest of the design features.

We view the process carried out in the project as a *collective creativity* process. *Collective creativity* is a term coined to describe the phenomenon where concepts and understanding emerge in people's mind through interacting with knowledge in the world; with external representations, with other people, or with computer systems [7].

Both the interaction designer and the programmer of the ART project exhibit creative processes by using external resources and representations generated by each other. Representations that the interaction designer generates and interacts-with are primarily sketches of visual appearances of systems to be created. He also uses application fragments that are programmed by the programmer, and other application programs that are commercially available or available as shareware. The programmer not only uses sketches produced by the designer, but also resorts to tangible physical artifacts, such as sculptures and buildings, as well as interactive demos programmed by other people.

The following section describes in detail what types of representations are used for what purposes by each of the two stakeholders in the ART project.

A Four-Faceted Model

In order to study the project as characterized above, we must have a framework to analyze the evolutionary collective creative process, which is partially inspired, partially structured, and partially situated.

There have been a number of models proposed to specify a process of creative activities. For instance, Csiksentmihalyi [2] represents a creative process consisting of collection, incubation, insight, and evaluation. A creative person collects information that may or may not be relevant to the task, incubates the information that have been accumulated with a specific goal of the task in mind, obtains insight as a creative leap, and evaluates the acquired insight to make sure that the result is innovative and useful. Although not imposed, the model presupposes a linear progression with an emphasis on the importance of "insight," which takes place within a creative person's mind. The model is more appropriate for specifying revolutionary, historical creativity of individuals rather than evolutionary ones where a number of small steps made through a cycle matter.

Shneiderman proposes the GENEX model for an evolutionary creative process, which consists of the following four steps [12]:

- collection: to learn from previous works and artifacts stored in the world;
- relation: to consult with peers and mentors at various stages of the process;
- creation: to explore, compose, and evaluate possible solutions; and
- donation: to disseminate the results in the world.

This model emphasizes the importance of externalization and communication with other people, and captures an evolutionary creative process of a person who demonstrates individual creativity. This model, however, is not framed to stress the interplay among stakeholders as a whole.

In our case study, the goal has been to analyze not an individual creative process of each stakeholder, but a collective creative process, which emerges through both individual work and collaboration between the designer and the programmer. We have to focus more on the *relating* and *donating* steps in the GENEX model to see how each stakeholder affects each other's design decisions leading to a creative artifact as a collective result.

To serve for this purpose, we have developed a model to represent a process of evolutionary collective creativity. The model has the following four facets (Figure 7): collection, representation, interaction, and reflection.

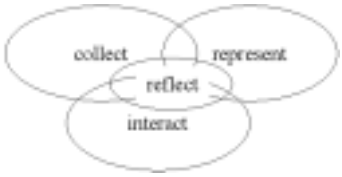

Figure 7: The Four-Faceted Model for Individuals Engaged in Collective Creativity

In this model, each stakeholder of the group, who demonstrate collective creativity, collect external information from outside resources. At the same time, each of them generates external representation, not necessarily as a creative artifact but as a medium to interact-with through the reflection-in-action process [10][11]. Reflection plays the central role integrating the other three aspects.

This model does not presuppose a linear process. Rather, it views a process as going back and forth among the four facets. Multiple facets may occur simultaneously; it is impossible to distinguish one type of activity from another in a definitive manner.

THE CREATIVE PROCESS DEMONSTRATED IN THE ART PROJECT

Based on the model described in the previous section, we have analyzed the evolutionary collective creative process

of the ART project. This section shows a result of our case study by basing our viewpoint on the use of representations throughout the process: what has been *collected*, *represented*, *interacted*-with, and *reflected*-on (1) by the interaction designer in the IW phase, (2) by the programmer in the IW phase, and (3) by the two of them in the FC phase of the process.

Representations Used by the Interaction Designer

In order to get ideas for visual interaction design, the interaction designer spends quite a long time in the collection process. In the same way as graphic designers create their own scrapbooks by clipping printed images and graphics and constantly browse them, or architects clip pictures of buildings and hang them on the wall, the interaction designer needs to be surrounded by a large amount of resourceful artifacts (Figure 8).

One type of mostly used such artifacts by the interaction designer is application software programs that are available as commercial products or as shareware available through the Web. By actually using and interacting with such programs, the interaction designer explores them from perspectives such as:

- what is an underlying intention embedded within the program;
- how the intention is mapped to tangible functionality;
- how interaction is implemented to communicate the functionality with the user; and
- whether the intention is critical for *his* interaction design.

For instance, in designing visual interaction for a user to author a hierarchical structure, the interaction designer interacted with a number of file-browser programs provided for Windows, Mac, and Linux; including Windows Explore, UNIX C-Shell, iPod, and coela for Mac. While interacting with the programs, the interaction designer examined aspects such as how to deal with the absolute path and how to navigate through the hierarchical structure. While doing so, the interaction designer gradually identified design intentions that were relevant to his design task.

The interaction designer also collects *experiences* of interacting with physical artifacts, including pencils, tables, chairs, or pairs of scissors. Although visual interaction design deals with representations within the computer world, the designer needs to understand the nature of physical world and how people interact with the real world. Interacting with physical objects helps the designer identify what are critical aspects and constraints of the real world and how people make use of them.

Through the processes of interacting with external representations, the interaction designer generates a number of sketches as designed artifacts. Some of the sketches are shown to the programmer as a part of design requirements, but some of the sketches are simply kept as a reminder for the designer himself for later reflection. Those sketches that are given to the programmer are the ones that are plausible to be programmable. Those sketches that are not shown to the programmer but kept as indices for the designer's thoughts are the ones that do not seem to be possible with the current computational power and resources, but would be possible in a near future when the power of hardware and processing speed improve. Thus, the designer *donates* his sketches for his future design task (see Figure 8).

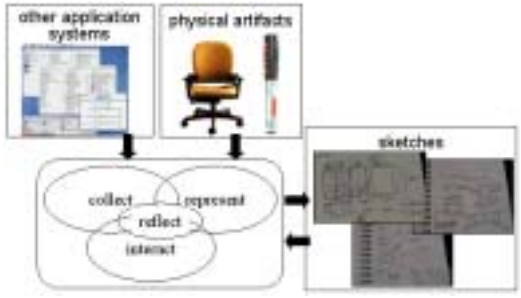

Figure 8: Representations Used by the Interaction Designer

Representations Used by the Programmer

The programmer mostly uses other open source programs written by other people as an external source of information. He learns what object-oriented models are used as an underlying architecture and how it is implemented (Figure 9).

In the same manner, the programmer attends to other representations, such as sculptures and buildings. By looking at such physical representations, the programmer tries to understand how they are built; the programmer resorts to the representation to explore possible processes of creating an artifact. When looking at an old pagoda in Japan, for instance, the programmer tries to understand in what order which parts of the pagoda have been assembled. Based on this interaction with the representation, the programmer makes an assumption of the way to create an artifact and validates the assumption by actually implementing it.

His assumption sometimes turns out to be wrong or insufficient to implement such an intended artifact. Different from visual interaction design generated by the interaction design in the form of sketches, programs can be "right" or "wrong." The program does not run properly if implemented in an inappropriate manner. "Wrong" programs do not necessarily mean that they are buggy. They are sometimes "wrong" because they do not run fast enough, or they take too much memory space. When the programs he produced turn out to be "wrong" in this manner, the programmer stores what he has generated: the underlying model and actual source code. Such programs would be "right" when the hardware performance

improves. Or his programs might run properly in a different context.

Thus, in the same way as designer saves his sketches for future references, the programmer saves his programs for future usage. He saves programmed source code for future reuse. In the ART project, we have observed that the programmer reused his program that he wrote five years ago. Donation process works for himself in a longer period of time (see Figure 9).

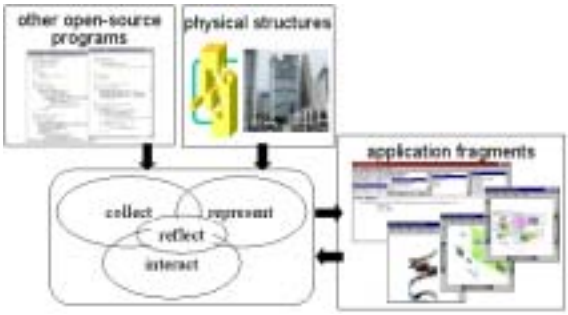

Figure 9: Representations Used by the Programmer

Representations Used in Collaboration

As we described above, design requirements are communicated through sketches. Application fragments, which are executable software programs, also serve as communicative media between the designer and the programmer as illustrated in Figures 5 and 6. Figure 11 shows examples of the application fragments that have been produced during the ART project.

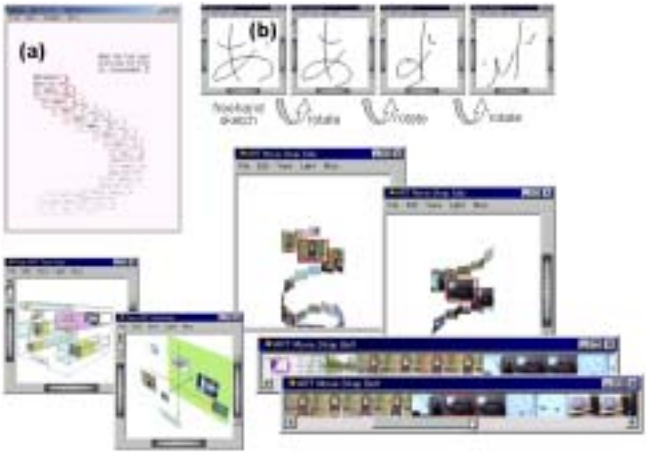

Figure 10: Application Fragments Produced by the ART Project

We have identified four interesting aspects in the use of such application fragments as communication media.

(1) To instantiate sketched design features into tangible forms. Application fragments are often used to instantiate design features sketched by the interaction designer. When the designer shows sketches to the programmer for the first time for a new design goal, the programmer tries to produce executable objects, which looks similar to the sketches, as quick as possible. This is necessary because the designer needs to actually see the object on the screen instead of on paper.

Once the designer sees the object and gives a *go* for the design, the programmer starts producing more robust underlying object models, and re-implements the interface.

While designing the object model, the programmer tries to make the model as rich and as flexible as possible so that he can respond to the emerging designer's requests in tuning and adjusting the design. Thus, the focus on the visual interaction design changes the way in programming software.

(2) To examine what is computationally feasible. The programmer implements application fragments to examine how much CPU power and memory is required for a certain design feature and how acceptable in the designer's eye the computer's response time is under regular computer settings. For instance, in Figure11-(a), a set of residue images of a text object as a trajectory in a 2-D space is implemented to see how many frames should be produced while the user drags the object in the space. Displaying too many residue images will require too much CPU power slowing down the display speed. The programmer and the designer had to collaboratively make a decision by making the compromise between what was desired and what was computationally feasible by actually interacting with the prototyped application fragment.

(3) To demonstrate what is computationally possible. The programmer implements application fragments to show what can be done with the current computational power even if they are not asked by the interaction designer. For instance, Figure 11-(b) shows a freehand-note application fragment. With this application fragment, while a user freehand-sketches in a window, the system keeps tracks of time taken for each stroke. When the user finishes and invokes the 3D view, the system converts the 2D sketch into a 3D model, with the time spent as the Z-depth. By rotating the view, the user can see in what order and in which speed the user has drawn the sketch.

The programmer has implemented this application fragment to show the designer how the user's freehand stroke is temporary traced. By interacting with the application fragment, the interaction designer has started thinking about what would be possible by having this kind of functionality, for instance, how the time could be mapped and represented in freehand applications.

(4) To develop common references. During the time when the designer and the programmer are focusing on a particular design requirement, they usually start using nicknames for some aspects of the envisioned system. They do not refer critical elements of the interface by "dry" names or functionality labels, such as "*the second scrollbar.*" Instead, they refer to them by nicknames, such

as "*the tiny little boy*" or "*the elder brother*" (translated from Japanese). These nicknames seem to play a critical role to make sure that they are referring to the same, critical aspect of the interface, and at the same time, the way the nickname is chosen helps them communicate implied meanings associated with the naming.

SUMMARY

This paper reports our case study on the ART project, the visual interaction design-centered software development project viewing as carrying out an evolutionary collective creative process. The interaction designer and the programmer collectively produce creative artifacts by communicating through external representations, including sketches and application fragments. Each of the two stakeholders uses outside resources for their creative activity, such as other interactive systems, physical artifacts, and programs. Both the designer and the programmer save their generated representations even if they are not useful for the current task. They "donate" such representations for their later creative tasks.

ACKNOWLEDGMENTS

We thank Yunwen Ye, Hiroko Asaoka, and Akio Takashima for their help in carrying out the ART project, conducting the case study, and in analyzing the result.

REFERENCES

1. Arias, E., Eden, H., Fischer, G., Gorman, A., Scharff, E., Transcending the Individual Human Mind - Creating Shared Understanding through Collaborative Design, ACM Transactions on Computer-Human Interaction, Vol.7, No.1, pp.84-113, March, 2000.
2. Csikszentmihalyi, M. and Sawyer, K. Creative Insight: The Social Dimension of a Solitary Moment, in The Nature of Insight, R.J. Sternberg and J.E. Davidson (eds.), MIT Press, Cambridge, MA, 1995.
3. Do, Y-L., and M.D. Gross. Inferring Design Intentions from Sketches: An Investigation of Freehand Drawing Conventions in Design, CAADRIA '97 - Proceedings of the Second Conference on Computer Aided Architectural Design Research in Asia, Liu, Y.-t., Tsou, J.Y., Hou, J.-H. (Eds.), Hu's Publishers, Taipei, 1997.
4. Fischer, G., and Nakakoji, K. Computational Environments Supporting Creativity in the Context of Lifelong Learning and Design, Knowledge-Based Systems Journal 10, 1 (June 1997), Elsevier Science Publishers, Amsterdam, the Netherlands, 21-28.
5. Lewis, S. The Art and Science of Smalltalk, Prentice Hall, 1995.
6. McLaughlin, S. and Gero, J.S. Creative Processes - Can They Be Automated? in Modeling Creativity and Knowledge-Based Creative Design (Reprints of the International Round-Table Conference: Modeling Creativity and Knowledge-Based Creative Design, Heron Island, December 1989), 69-94.
7. Nakakoji, K., Ohira, M., Yamamoto, Y., Computational Support for Collective Creativity, Knowledge-Based Systems Journal, Elsevier Science, Vol.13, No.7-8, pp.451-458, December, 2000.
8. Nakakoji, K. Yamamoto, Y., Reeves, B.N., Takada,S.,Two-Dimensional Positioning as a Means for Reflection in Design, Design of Interactive Systems (DIS'2000), ACM, New York, NY, pp. 145-154, August, 2000.
9. Nakakoji, K., Yamamoto, Y., Aoki, A., Third Annual Special Issue on Interface Design, Interactions, ACM Press, Vol.IX.2, pp.99-102, March+April, 2002.
10. Schoen, D A, The Reflective Practitioner: How Professionals Think in Action, Basic Books, NY, 1983.
11. Sharples, M. Cognitive Support and the Rhythm of Design, Artificial Intelligence and Creativity, Dartnall T. (Ed.), Kluwer Academic Publishers, the Netherlands, 385-402, 1994.
12. Shneiderman, B., Creating Creativity: User Interfaces for Supporrting Innovation, ACM Transactions on Computer-Human Interaction, Vol.7, No.1, pp.114-138, March, 2000.
13. Weed, B., Visual Interaction Design: The Industrial Design of the Software Industry, ACM SIGCHI Bulletin Vol.28 No.3, July 1996.
14. Yamamoto, Y., Nakakoji, K., Takada, S. Hands-on Representations in a Two-Dimensional Space for Early Stages of Design, Knowledge-Based Systems Journal, Elsevier Science, Vol.13, No.6, pp.375-384, November, 2000.
15. Yamamoto, Y., Amplifying Representational Talkback: Interactive Systems Using Spatial Positioning to Support Early Stages of Information Design, Doctoral Dissertation, Graduate School of Information Science, Nara Institute of Science and Technology, Japan, March, 2001.
16. Yamamoto, Y., Nakakoji, K., Aoki, A., Visual Interaction Design for Tools to Think with: Interactive Systems for Designing Linear Information, Proceedings of the Working Conference on Advanced Visual Interfaces (AVI2002), ACM Press, Torento, Italy, pp.367-372, May, 2002.
17. Yamamoto, Y., Nakakoji, K., Aoki, A., Spatial Hypertext for Linear-Information Authoring: Interaction Design and System Development Based on the ART Design Principle, Proceedings of Hypertext2002, ACM Press, pp. 35-44, June, 2002.
18. Zhang, J, The Nature of External Representations in Problem Solving, in: Cognitive Science, 21(2), pp.179-217, 1997.